

INTRODUCTION TO ALGORITHMS

FOURTH EDITION

LECTURE NOTES FOR CHAPTER 3

Characterizing Running Times

CHAPTER 3 OVERVIEW

Goals

- A way to describe behavior of functions in the limit. We’re studying asymptotic efficiency.
- Describe growth of functions.
- Focus on what’s important by abstracting away low-order terms and constant factors.
- How we indicate running times of algorithms.
- A way to compare “sizes” of functions:

$$\begin{array}{lll} O & \approx & \leq \\ \Omega & \approx & \geq \\ \Theta & \approx & = \\ o & \approx & < \\ \omega & \approx & > \end{array}$$

O-notation

O-notation characterizes an *upper bound* on the asymptotic behavior of a function: it says that a function grows *no faster* than a certain rate. This rate is based on the highest order term.

For example:

$f(n) = 7n^3 + 100n^2 - 20n + 6$ is $O(n^3)$, since the highest order term is $7n^3$, and therefore the function grows no faster than n^3 .

The function $f(n)$ is also $O(n^5)$, $O(n^6)$, and $O(n^c)$ for any constant $c \geq 3$.

Ω -notation

Ω -notation characterizes a *lower bound* on the asymptotic behavior of a function.

For example:

$f(n) = 7n^3 + 100n^2 - 20n + 6$ is $\Omega(n^3)$, since the highest-order term, n^3 , grows at least as fast as n^3 .

The function $f(n)$ is also $\Omega(n^2)$, $\Omega(n)$ and $\Omega(n^c)$ for any constant $c \leq 3$.

Θ -notation

Θ -notation characterizes a *tight bound* on the asymptotic behavior of a function: it says that a function grows *precisely* at a certain rate, again based on the highest-order term.

If a function is both $O(f(n))$ and $\Omega(f(n))$, then a function is $\Theta(f(n))$.

EXAMPLE: INSERTION-SORT

INSERTION-SORT(A, n)

```
1  for  $i = 2$  to  $n$ 
2       $key = A[i]$ 
3      // Insert  $A[i]$  into the sorted subarray  $A[1 : i - 1]$ .
4       $j = i - 1$ 
5      while  $j > 0$  and  $A[j] > key$ 
6           $A[j + 1] = A[j]$ 
7           $j = j - 1$ 
8       $A[j + 1] = key$ 
```

EXAMPLE: INSERTION-SORT (continued)

First, show that INSERTION-SORT runs in $O(n^2)$ time, regardless of the input:

- The outer **for** loop runs $n - 1$ times regardless of the values being sorted.
- The inner **while** loop iterates at most $i - 1$ times.
- The exact number of iterations the **while** loop makes depends on the values it iterates over, but it will definitely iterate between 0 and $i - 1$ times.
- Since i is at most n , the total number of iterations of the inner loop is at most $(n - 1)(n - 1)$, which is less than n^2 .

Each inner loop iteration takes constant time, for a total of at most cn^2 for some constant c , or $O(n^2)$.

EXAMPLE: INSERTION-SORT (continued)

Now show that INSERTION-SORT has a worst-case running time of $\Omega(n^2)$:

- Observe that for a value to end up k positions to the right of where it started, the line $A[j + 1] = A[j]$ must have been executed k times.
- Assume that n is a multiple of 3 so that we can divide the array A into groups of $n/3$ positions.

$A[1 : n/3]$	$A[n/3 + 1 : 2n/3]$	$A[2n/3 + 1 : n]$
each of the $n/3$ largest values moves	through each of these $n/3$ positions	to somewhere in these $n/3$ positions



EXAMPLE: INSERTION-SORT (continued)

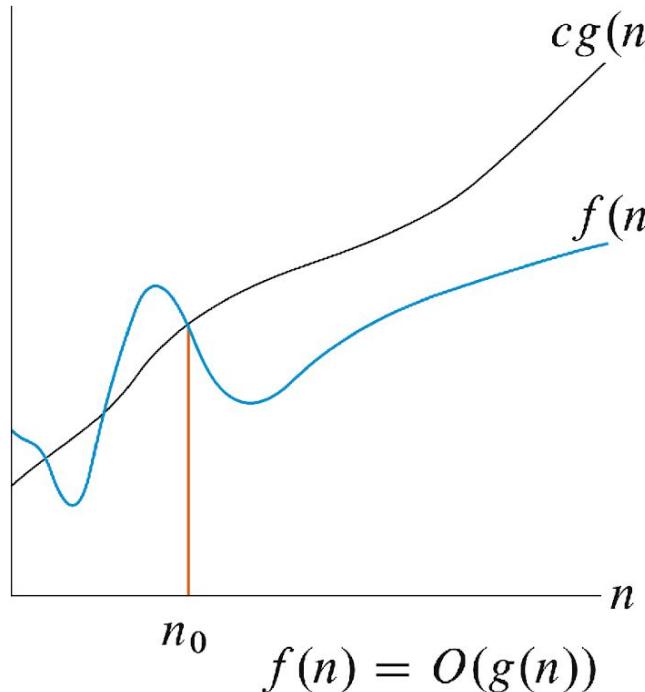
Because at least $n/3$ values must pass through at least $n/3$ positions, the line $A[j + 1] = A[j]$ executes at least $(n/3)(n/3) = n^2/9$ times, which is $\Omega(n^2)$. For this input, INSERTION-SORT takes time $\Omega(n^2)$.

Since we have shown that INSERTION-SORT runs in $O(n^2)$ time in all cases and that there is an input that makes it take $\Omega(n^2)$ time, we can conclude that the worst-case running time of INSERTION-SORT is $\Theta(n^2)$.

The constant factors for the upper and lower bounds may differ. That doesn't matter.

O -notation

$O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that}$
 $0 \leq f(n) \leq cg(n) \text{ for all } n \geq n_0\}$.



$g(n)$ is an *asymptotic upper bound* for $f(n)$.
If $f(n) \in O(g(n))$, we write $f(n) = O(g(n))$

Example

$2n^2 = O(n^3)$, with $c = 1$ and $n_0 = 2$.

Examples of functions in $O(n^2)$:

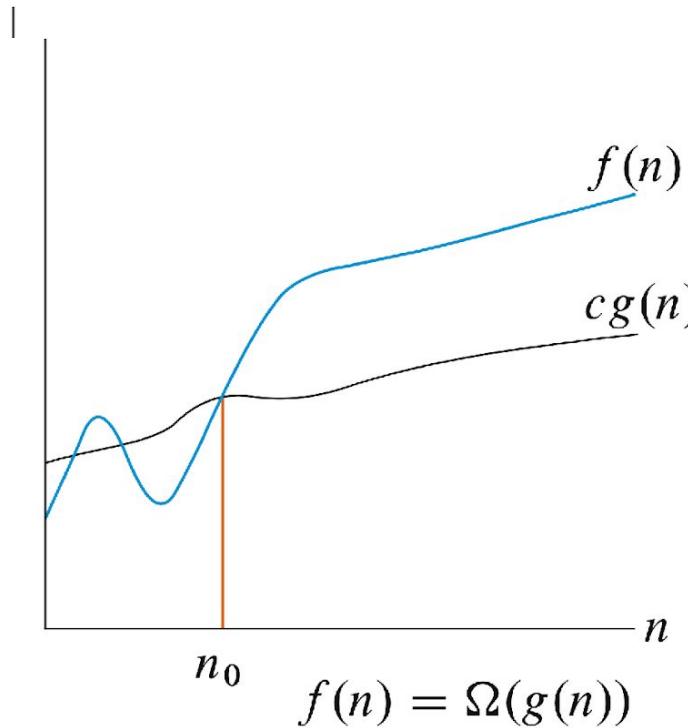
$$\begin{aligned} & n^2 \\ & n^2 + n \\ & n^2 + 1000n \\ & 1000n^2 + 1000n \end{aligned}$$

Also,

$$\begin{aligned} & n \\ & n/1000 \\ & n^{1.99999} \\ & n^2 / \lg \lg \lg n \end{aligned}$$

Ω -notation

$\Omega(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that}$
 $0 \leq cg(n) \leq f(n) \text{ for all } n \geq n_0\}$.



$g(n)$ is an *asymptotic lower bound* for $f(n)$.

Example

$\sqrt{n} = \Omega(\lg n)$, with $c = 1$ and $n_0 = 16$.

Examples of functions in $\Omega(n^2)$:

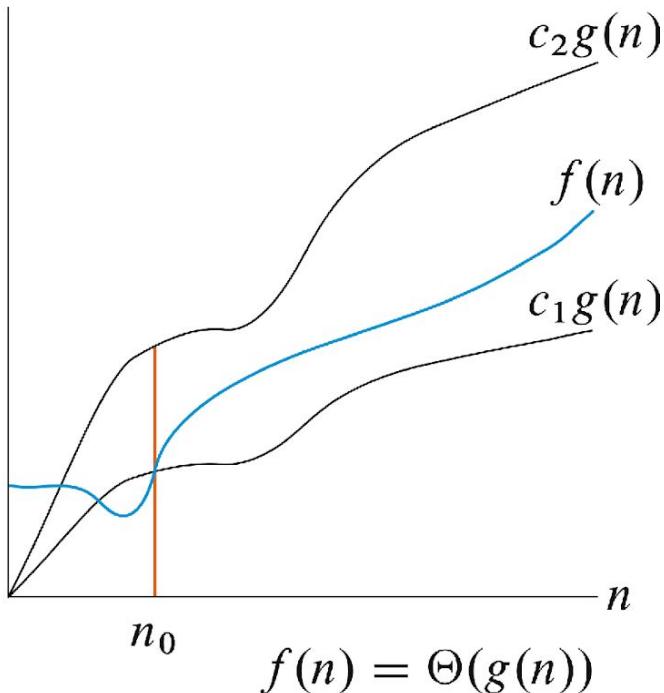
$$\begin{aligned} & n^2 \\ & n^2 + n \\ & n^2 - n \\ & 1000n^2 + 1000n \\ & 1000n^2 - 1000n \end{aligned}$$

Also,

$$\begin{aligned} & n^3 \\ & n^{2.00001} \\ & n^2 \lg \lg \lg n \\ & 2^{2^n} \end{aligned}$$

Θ -notation

$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that}$
 $0 \leq c_1g(n) \leq f(n) \leq c_2g(n) \text{ for all } n \geq n_0\}$.



Example

$n^2/2 - 2n = \Theta(n^2)$, with $c_1 = 1/4$, $c_2 = 1/2$, and $n_0 = 8$.

Theorem

$f(n) = \Theta(g(n))$ if and only if $f = O(g(n))$ and $f = \Omega(g(n))$.

Leading constants and low-order terms don't matter.

$g(n)$ is an *asymptotically tight bound* for $f(n)$.

ASYMPTOTIC NOTATION AND RUNNING TIMES

Need to be careful to use asymptotic notation correctly when characterizing a running time. Asymptotic notation describes functions, which in turn describe running times. Must be careful to specify *which* running time.

For example:

The worst-case running time for insertion sort is $O(n^2)$, $\Omega(n^2)$, and $\Theta(n^2)$; all are correct. Prefer to use $\Theta(n^2)$ here, since it's the most precise.

The best-case running time for insertion sort is $O(n)$, $\Omega(n)$, and $\Theta(n)$; prefer $\Theta(n)$.

ASYMPTOTIC NOTATION AND RUNNING TIMES (continued)

But *cannot* say that the running time for insertion sort is $\Theta(n^2)$, with “worst-case” omitted. Omitting the case means making a blanket statement that covers *all* cases, and insertion sort does *not* run in $\Theta(n^2)$ time in all cases.

Can make the blanket statement that the running time for insertion sort is $O(n^2)$, or that it’s $\Omega(n)$, because these asymptotic running times are true for all cases.

For merge sort, its running time is $\Theta(n \lg n)$ in all cases, so it’s OK to omit which case.

ASYMPTOTIC NOTATION AND RUNNING TIMES

(continued)

Common error: conflating O -notation with Θ -notation by using O -notation to indicate an asymptotically tight bound. O -notation gives only an asymptotic upper bound. Saying “an $O(n \lg n)$ -time algorithm runs faster than an $O(n^2)$ -time algorithm” is not necessarily true. An algorithm that runs in $\Theta(n)$ time also runs in $O(n^2)$ time. If you really mean an asymptotically tight bound, then use Θ -notation.

Use the simplest and most precise asymptotic notation that applies. Suppose that an algorithm’s running time is $3n^2 + 20n$. Best to say that it’s $\Theta(n^2)$. Could say that it’s $O(n^3)$, but that’s less precise. Could say that it’s $\Theta(3n^2 + 20n)$ but that obscures the order of growth.

ASYMPTOTIC NOTATION IN EQUATIONS

When on right-hand side:

$O(n^2)$ stands for some anonymous function in the set $O(n^2)$.

$2n^2 + 3n + 1 = 2n^2 + \Theta(n)$ means $2n^2 + 3n + 1 = 2n^2 + f(n)$ for some $f(n) \in \Theta(n)$. In particular, $f(n) = 3n + 1$.

When on left-hand side:

No matter how the anonymous functions are chosen on the left-hand side, there is a way to choose the anonymous functions on the right-hand side to make the equation valid.

Interpret $2n^2 + \Theta(n) = \Theta(n^2)$ as meaning *for all* functions $f(n) \in \Theta(n)$, there exists a function $g(n) \in \Theta(n^2)$ such that $2n^2 + f(n) = g(n)$.

ASYMPTOTIC NOTATION IN EQUATIONS

(continued)

Can chain together:

$$\begin{aligned} 2n^2 + 3n + 1 &= 2n^2 + \Theta(n) \\ &= \Theta(n^2). \end{aligned}$$

Interpretation:

- First equation: There exists $f(n) \in \Theta(n)$ such that $2n^2 + 3n + 1 = 2n^2 + f(n)$.
- Second equation: For all $g(n) \in \Theta(n)$ (such as the $f(n)$ used to make the first equation hold), there exists $h(n) \in \Theta(n^2)$ such that $2n^2 + g(n) = h(n)$.

SUBTLE POINT: ASYMPTOTIC NOTATION IN RECURRENCES

Often abuse asymptotic notation when writing recurrences: $T(n) = O(1)$ for $n < 3$. Strictly speaking, this statement is meaningless. Definition of O -notation says that $T(n)$ is bounded above by a constant $c > 0$ for $n \geq n_0$, for some $n_0 > 0$. The value of $T(n)$ for $n < n_0$ might not be bounded. So when we say $T(n) = O(1)$ for $n < 3$, cannot determine any constraint on $T(n)$ when $n < 3$ because could have $n_0 > 3$.

What we really mean is that there exists a constant $c > 0$ such that $T(n) \leq c$ for $n < 3$. This convention allows us to avoid naming the bounding constant so that we can focus on the more important part of the recurrence.

O-notation

$o(g(n)) = \{f(n) : \text{for all constants } c > 0, \text{there exists a constant } n_0 > 0 \text{ such that } 0 \leq f(n) < cg(n) \text{ for all } n \geq n_0\}$.

Another view, probably easier to use: $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$.

$$n^{1.9999} = o(n^2)$$

$$n^2 / \lg n = o(n^2)$$

$$n^2 \neq o(n^2) \text{ (just like } 2 \neq 2)$$

$$n^2/1000 \neq o(n^2)$$

ω -notation

$\omega(g(n)) = \{f(n) : \text{for all constants } c > 0, \text{ there exists a constant } n_0 > 0 \text{ such that } 0 \leq cg(n) < f(n) \text{ for all } n \geq n_0\}$.

Another view, again, probably easier to use: $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty$.

$$n^{2.0001} = \omega(n^2)$$

$$n^2 \lg n = \omega(n^2)$$

$$n^2 \neq \omega(n^2)$$

COMPARISONS OF FUNCTIONS

Transitivity:

$$f(n) = \Theta(g(n)) \text{ and } g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n)).$$

Same for O , Ω , o , and ω .

Reflexivity:

$$f(n) = \Theta(f(n)).$$

Same for O and Ω .

Symmetry:

$$f(n) = \Theta(g(n)) \text{ if and only if } g(n) = \Theta(f(n)).$$

Transpose symmetry:

$$f(n) = O(g(n)) \text{ if and only if } g(n) = \Omega(f(n)).$$

$$f(n) = o(g(n)) \text{ if and only if } g(n) = \omega(f(n)).$$

COMPARISONS OF FUNCTIONS (continued)

Comparisons:

- $f(n)$ is *asymptotically smaller* than $g(n)$ if $f(n) = o(g(n))$.
- $f(n)$ is *asymptotically larger* than $g(n)$ if $f(n) = \omega(g(n))$.

No trichotomy. Although intuitively, we can liken O to \leq , Ω to \geq , etc., unlike real numbers, where $a < b$, $a = b$, or $a > b$, we might not be able to compare functions.

Example: $n^{1+\sin n}$ and n , since $1 + \sin n$ oscillates between 0 and 2.

LOGARITHMS

Notations:

$$\lg n = \log_2 n \quad (\text{binary logarithm}) ,$$

$$\ln n = \log_e n \quad (\text{natural logarithm}) ,$$

$$\lg^k n = (\lg n)^k \quad (\text{exponentiation}) ,$$

$$\lg \lg n = \lg(\lg n) \quad (\text{composition}) .$$

Logarithm functions apply only to the next term in the formula, so that $\lg n + k$ means $(\lg n) + k$, and *not* $\lg(n + k)$.

In the expression $\log_b a$:

- Hold b constant \Rightarrow the expression is strictly increasing as a increases.
- Hold a constant \Rightarrow the expression is strictly decreasing as b increases.

LOGARITHMS (continued)

$$a = b^{\log_b a},$$

$$\log_c(ab) = \log_c a + \log_c b,$$

$$\log_b a^n = n \log_b a,$$

$$\log_b a = \frac{\log_c a}{\log_c b},$$

$$\log_b(1/a) = -\log_b a,$$

$$\log_b a = \frac{1}{\log_a b},$$

$$a^{\log_b c} = c^{\log_b a}.$$